***INDEX***

Indexing is a way of sorting a number of records on multiple fields. Creating an index on a field in a table creates another data structure which holds the field value, and pointer to the record it relates to. This index structure is then sorted, allowing Binary Searches to be performed on it.

The downside to indexing is that these indexes require additional space on the disk, since the indexes are stored together in a table using the MyISAM engine, this file can quickly reach the size limits of the underlying file system if many fields within the same table are indexed.

**What is an Index?**

***INDEX:->***An index is a small table having only two columns. The first column contains a copy of the primary or candidate key of a table and the second column contains a set of pointers holding the address of the disk block where that particular key value can be found.

The ***advantage*** of using index lies in the fact is that

index makes search operation perform very fast.

***EXAMPLE:->***

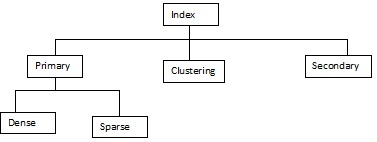
Suppose a table has a several rows of data, each row is 20 bytes wide. If you want to search for the record number 100, the management system must thoroughly read each and every row and after reading 99x20 = 1980 bytes it will find record number 100. If we have an index, the management system starts to search for record number 100 not from the table, but from the index. The index, containing only two columns, may be just 4 bytes wide in each of its rows. After reading only 99x4 = 396 bytes of data from the index the management system finds an entry for record number 100, reads the address of the disk block where record number 100 is stored and directly points at the record in the physical storage device. The result is a much quicker access to the record (a speed advantage of 1980:396).

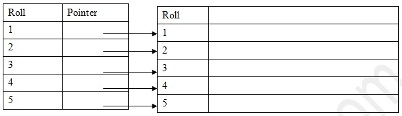
The only minor d***isadvantage*** of using index is that it takes up a little more space than the main table.

Additionally, index needs to be updated periodically for insertion or deletion of records in the main table.

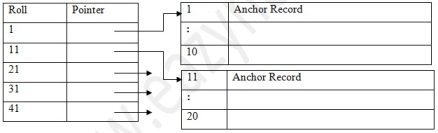
However, the advantages are so huge that these disadvantages can be considered negligible.

**Types of Index**



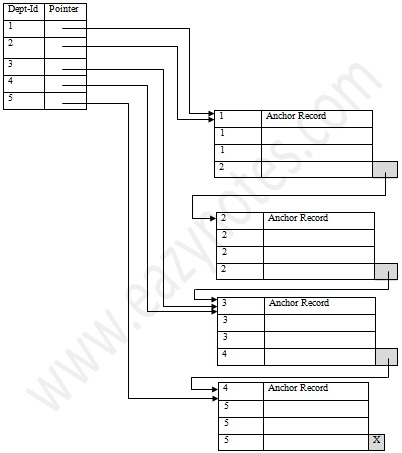
***1. Primary Index***In primary index, there is a one-to-one relationship between the entries in the index table and the records in the main table. Primary index can be of two types:  
  
***1.1 Dense primary index***: the number of entries in the index table is the same as the number of entries in the main table. In other words, each and every record in the main table has an entry in the index.  
  


***1.2 Sparse or Non-Dense Primary Index:***For large tables the Dense Primary Index itself begins to grow in size. To keep the size of the index smaller, instead of pointing to each and every record in the main table, the index points to the records in the main table in a gap. See the following example.



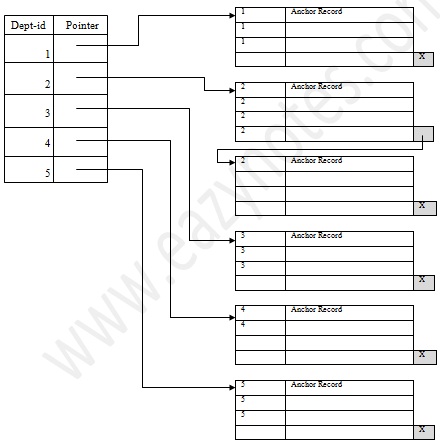
As you can see, the data blocks have been divided in to several blocks, each containing a fixed number of records (in our case 10). The pointer in the index table points to the first record of each data block, which is known as the Anchor Record for its important function. If you are searching for roll 14, the index is first searched to find out the highest entry which is smaller than or equal to 14. We have 11. The pointer leads us to roll 11 where a short sequential search is made to find out roll 14.

***2. Clustering Index***It may happen sometimes that we are asked to create an index on a non-unique key, such as Dept-id. There could be several employees in each department. Here we use a clustering index, where all employees belonging to the same Dept-id are considered to be within a single cluster, and the index pointers point to the cluster as a whole.

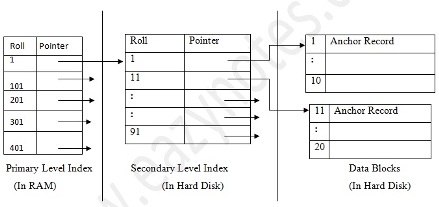
******

Let us explain this diagram. The disk blocks contain a fixed number of records (in this case 4 each). The index contains entries for 5 separate departments. The pointers of these entries point to the anchor record of the block where the first of the Dept-id in the cluster can be found. The blocks themselves may point to the anchor record of the next block in case a cluster overflows a block size. This can be done using a special pointer at the end of each block (comparable to the next pointer of the linked list organization).

The previous scheme might become a little confusing because one disk block might be shared by records belonging to different cluster. A better scheme could be to use separate disk blocks for separate clusters. This has been explained in the next page.



In this scheme, as you can see, we have used separate disk block for the clusters. The pointers, like before, have pointed to the anchor record of the block where the first of the cluster entries would be found. The block pointers only come into action when a cluster overflows the block size, as for Dept-id 2. This scheme takes more space in the memory and the disk, but the organization in much better and cleaner looking.

***3. Secondary Index***While creating the index, generally the index table is kept in the primary memory (RAM) and the main table, because of its size is kept in the secondary memory (Hard Disk). Theoretically, a table may contain millions of records (like the telephone directory of a large city), for which even a sparse index becomes so large in size that we cannot keep it in the primary memory. And if we cannot keep the index in the primary memory, then we lose the advantage of the speed of access. For very large table, it is better to organize the index in multiple levels. See the following example.  
  


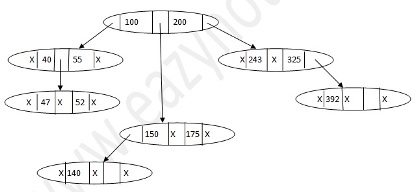
In this scheme, the primary level index, (created with a gap of 100 records, and thereby smaller in size), is kept in the RAM for quick reference. If you need to find out the record of roll 14 now, the index is first searched to find out the highest entry which is smaller than or equal to 14. We have 1. The adjoining pointer leads us to the anchor record of the corresponding secondary level index, where another similar search is conducted. This finally leads us to the actual data block whose anchor record is roll 11.   We now come to roll 11 where a short sequential search is made to find out roll 14.

**Multilevel Index**  
The Multilevel Index is a modification of the secondary level index system. In this system we may use even more number of levels in case the table is even larger.

**Index in a Tree like Structure**

We can use tree-like structures as index as well. For example, a binary search tree can also be used as an index. If we want to find out a particular record from a binary search tree, we have the added advantage of binary search procedure, that makes searching be performed even faster. A binary tree can be considered as a **2-way Search Tree**, because it has two pointers in each of its nodes, thereby it can guide you to two distinct ways. Remember that for every node storing 2 pointers, the number of value to be stored in each node is one less than the number of pointers, i.e. each node would contain 1 value each.

**M-Way Search Tree**   
The abovementioned concept can be further expanded with the notion of the m-Way Search Tree, where m represents the number of pointers in a particular node. If m = 3, then each node of the search tree contains 3 pointers, and each node would then contain 2 values.  
A sample m-Way Search Tree with m = 3 is given in the following.



----------------

Clustered index  
Non-clustered

Clustered index  
Clustered index exists as sorted row on disk.   
Clustered index re-orders the table record.   
Clustered index contains record in the leaf level of the B-tree.   
There can be only one Clustered index possible in a table.

***Non-clustered***  
Non-clustered index is the index in which logical order doesn’t match with physical order of stored data on disk.   
Non-clustered index contains index key to the table records in the leaf level.   
There can be one or more Non-clustered indexes in a table

-------------------

#### Types of indexes.

The types of indexes are

1***. Clustered:*** Clustered index sorts and stores the rows data of a table / view based on the order of clustered index key. Clustered index key is implemented in B-tree index structure.

***2. Nonclustered:*** A non clustered index is created using clustered index. Each index row in the non clustered index has non clustered key value and a row locator. Locator positions to the data row in the clustered index that has key value.

***3. Unique:*** Unique index ensures the availability of only non-duplicate values and therefore, every row is unique.

***4. Full-text:*** It supports is efficient in searching words in string data. This type of indexes is used in certain database managers.

***5. Spatial:*** It facilitates the ability for performing operations in efficient manner on spatial objects. To perform this, the column should be of geometry type.

***6. Filtered:*** A non clustered index. Completely optimized for query data from a well defined subset of data. A filter is utilized to predicate a portion of rows in the table to be indexed.

=============

Index is used to solve some query fast.  
  
we have 2 types of cursors.  
1 Bitmap Index ( used only on Low cardinality columns)  
  
2. Btree Index ( used only for High Cardinality of   
  
columns)

----------------------------

***Indexes*** are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically and are then referred to one or more specific page numbers.

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

## The CREATE INDEX Command:

The basic syntax of **CREATE INDEX** is as follows:

CREATE INDEX index\_name ON table\_name;

## *Single-Column Indexes:*

A single-column index is one that is created based on only one table column. The basic syntax is as follows:

CREATE INDEX index\_name

ON table\_name (column\_name);

## *Unique Indexes:*

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows:

CREATE UNIQUE INDEX index\_name

on table\_name (column\_name);

## *Composite Indexes:*

A composite index is an index on two or more columns of a table. The basic syntax is as follows:

CREATE INDEX index\_name

on table\_name (column1, column2);

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

## *Implicit Indexes:*

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

## *The DROP INDEX Command:*

An index can be dropped using SQL **DROP** command. Care should be taken when dropping an index because performance may be slowed or improved.

The basic syntax is as follows:

DROP INDEX index\_name;

You can check [INDEX Constraint](http://www.tutorialspoint.com/sql/sql-index.htm) chapter to see actual examples on Indexes.

## *When should indexes be avoided?*

Although indexes are intended to enhance a database's performance, there are times when they should be avoided. The following guidelines indicate when the use of an index should be reconsidered:

* Indexes should not be used on small tables.
* Tables that have frequent, large batch update or insert operations.
* Indexes should not be used on columns that contain a high number of NULL values.
* Columns that are frequently manipulated should not be indexed.

=================================

**Bitmap index**

A bitmap index is a special kind of index that stores the bulk of its data as [bit arrays](https://en.wikipedia.org/wiki/Bit_array) (bitmaps) and answers most queries by performing [bitwise logical operations](https://en.wikipedia.org/wiki/Bitwise_operation) on these bitmaps. The most commonly used indexes, such as [B+trees](https://en.wikipedia.org/wiki/B%2Btree), are most efficient if the values they index do not repeat or repeat a small number of times. In contrast, the bitmap index is designed for cases where the values of a variable repeat very frequently. For example, the gender field in a customer database usually contains at most three distinct values: male, female or unknown (not recorded). For such variables, the bitmap index can have a significant performance advantage over the commonly used trees.

**Dense index**

A dense index in [databases](https://en.wikipedia.org/wiki/Database) is a [file](https://en.wikipedia.org/wiki/Computer_file) with pairs of keys and [pointers](https://en.wikipedia.org/wiki/Pointer_(computer_programming)) for every [record](https://en.wikipedia.org/wiki/Record_(computer_science)) in the data file. Every key in this file is associated with a particular pointer to*a record* in the sorted data file. In clustered indices with duplicate keys, the dense index points *to the first record* with that key.[[3]](https://en.wikipedia.org/wiki/Database_index#cite_note-3)

**Sparse index**

A sparse index in databases is a file with pairs of keys and pointers for every [block](https://en.wikipedia.org/wiki/Block_(data_storage)) in the data file. Every key in this file is associated with a particular pointer *to the block* in the sorted data file. In clustered indices with duplicate keys, the sparse index points *to the lowest search key* in each block.

**Reverse index**

A reverse key index reverses the key value before entering it in the index. E.g., the value 24538 becomes 83542 in the index. Reversing the key value is particularly useful for indexing data such as sequence numbers, where new key values monotonically increase.